

**WEST****End of Result Set**☐ **Generate Collection** **Print**

L7: Entry 1 of 1

File: USPT

Feb 4, 2003

DOCUMENT-IDENTIFIER: US 6516356 B1

TITLE: Application interface to a media server and a method of implementing the sameAbstract Text (4):

The present invention further encompasses a common application program interface (API) which converts high-level generic commands received from a computer application into device-level commands which are output to a plurality of media devices including media servers which stores media objects. The common API includes a plurality of individual APIs which each perform a specific function.

US Patent No. (1):

6516356

Brief Summary Text (3):

The present invention relates to an application interface, and more particularly to an application interface to a media server and a method of implementing the same.

Brief Summary Text (5):

Conventional multimedia data storage systems are often employed to retain large amounts of multimedia data which are made available for multimedia applications. These conventional multimedia data storage systems may also incorporate a digital library, such as that which is described in U.S. Pat. No. 5,649,185 to Antognini et al. A problem that arises with conventional multimedia data storage systems stems from the fact that they often employ multiple media servers which deliver specific types of data to a client media output station such as a client viewer including a display. However, because each media server supports only its own device-level commands, it is difficult for applications running within the conventional multimedia data storage system to interact with multiple media servers. Accordingly, each application communicating with a particular media server must take into account the unique characteristics of that media server. However, such a scheme can become burdensome when there are numerous media servers. Moreover, such a scheme requires that applications be continuously updated when new media servers are incorporated into the multimedia data storage system.

Brief Summary Text (7):

It is an object of the present invention to provide a robust management scheme which provides a common interface to media servers.

Brief Summary Text (8):

It is another object of the present invention to provide a common interface to media servers which conceals the media server specific device commands from applications which interact with the media servers included within the system.

Brief Summary Text (12):

The present invention further encompasses a common application program interface (API) which converts a high-level generic command received from a computer application into one or more device-level commands which are output to a plurality of media devices including media servers which store media objects. The common API includes a plurality of individual APIs which each perform a specific function.

Brief Summary Text (13):

According to one aspect of the invention, the plurality of individual APIs comprise first and second groups of individual APIs. The first group of the individual APIs corresponds to a first group of member functions associated with a class defining objects which represent media servers. The second group of the individual APIs corresponds to a second group of member functions associated with a class defining objects which represent a logical description of a physical format of media objects.

Detailed Description Text (2):

The present invention is directed to a media manager, which is preferably implemented in a multimedia data storage system, that provides a common application program interface to media servers included within the system. The media manager incorporates a management scheme which facilitates communication between applications running within the multimedia data storage system and the media device(s).

Detailed Description Text (3):

FIG. 1 depicts a media manager 5 which receives high-level commands from a requesting application 8. The media manager 5 converts the high-level commands into device specific commands which are output to first through N-TH media devices 25, which may be a media archive or a media server for handling specific types of media data such as video or audio data. Using the media manager 5, the requesting application 8 can request that a specific function be performed by the media devices 25 without having to take into account the idiosyncracies of those media devices 25. In addition, the media manager 5 also shields the requesting application 8 from having to account for internal changes to the media devices 25.

Detailed Description Text (4):

The media manager 5 includes a high-level command processor 10 which receives the high-level commands from the requesting application 8 and identifies whether a command corresponds to a particular type of media server which the media manager 5 supports. If the high-level command processor 10 determines that the command issued by the requesting application 8 corresponds to a media device 25 supported by the media manager 5, the processor 10 passes the request to a device-specific code-mapping module 15 which handles the high-level commands destined for that media device 25.

Detailed Description Text (8):

The individual APIs correspond to member functions of at least two classes. The first class, SERVER, defines objects which represent the media servers 130 shown in FIG. 3. However, the SERVER class may be derived from a third class, MEDIA\_DEVICE, in order to accommodate both media servers and media archives. Thus, a fourth class, ARCHIVE, may be provided which defines objects that represent the media archives 140. However, discussion of the ARCHIVE class is omitted because the member functions which make up the class are similar to those provided in the SERVER class. A discussion of a media archive can be found in U.S. Application, entitled "A MULTIMEDIA DATA STORAGE SYSTEM INCLUDING A MEDIA SERVER AND MEDIA ARCHIVE AND A METHOD OF MANAGING THE MEDIA SERVER AS A CACHE DEVICE FOR THE MEDIA ARCHIVE", which is incorporated herein by reference and filed concurrently with the present application.

Detailed Description Text (11):

SERVER APPLICATION PROGRAM INTERFACES

Detailed Description Text (14):

A COPY\_MEDIA API is provided to copy a media object from one server to one or more other servers.

Detailed Description Text (15):

A DELETE API is provided in order to remove a media object from the server subsystem.

Detailed Description Text (20):

An INIT\_SERVER API is provided to initialize a media manager client library. However, the INIT\_SERVER API must be the first API that the requesting application

calls before requesting another media manager service.

Detailed Description Text (21):

A LIST\_MEDIA\_GROUPS API is provided to obtain a list of configured media group names.

Detailed Description Text (22):

A LIST\_MEDIA\_NAMES API is provided to obtain a list of names of media objects associated with a particular media group of media objects.

Detailed Description Text (24):

An OPEN\_SESSION API is provided to open a session from the application to the video server subsystem.

Detailed Description Text (26):

A C\_SERVER API is provided as a constructor to create an instance of the SERVER class for a specific media server type.

Detailed Description Text (27):

A D\_SERVER API is provided as a default destructor to destroy an instance of the SERVER class for a specific media server type.

Detailed Description Text (30):

A DESTAGE API is provided to destage (i.e. transfer) a media object from a media server to a media archive.

Detailed Description Text (35):

A RETRIEVE API is provided to retrieve a media object from a media server. The RETRIEVE API is used in conjunction with the GET\_META\_DATA API.

Detailed Description Text (40):

The operation of the media manager 5 will now be discussed in connection with FIG. 2. In step 50, a high-level command is received by the media manager 5 from the requesting application 8 along with a target media device. Thereafter, in step 55, the video server type is determined by the high-level command processor 10 of the media manager 5 based on the information provided by the requesting application 8. Subsequently, in step 60, the request for the specific machine type is validated by the high-level command processor 10.

Detailed Description Text (42):

An exemplary multimedia data storage system which incorporates the inventive media manager shown in FIG. 1 is described below with reference to FIGS. 3 and 4 in which the common API of the present invention is used in a digital library environment. The multimedia data storage system includes a library client 100, a client media output station 110, a digital library 120, media servers 130 and a media archive 140 which are implemented in a computer based system. The relationship between the client 100 and the digital library 120 is explained in greater detail below with reference to FIG. 4 which also shows the internal elements of the digital library 120. The internal structure of the digital library 120 is further described in U.S. Pat. No. 5,649,185, as noted above.

Detailed Description Text (43):

As shown in FIG. 4, the digital library 120 includes a library server 210 and at least one media object server 220 which interact with the library client 100. In addition, a communications isolator is provided in the digital library which allows the library server 210, the media object server 220 and the library client 100 to communicate with one another without concern for complex communications protocols. The library server 210, the media object server 220 and the library client 100 are connected by a communications network, such as a local area network (LAN) or a wide area network (WAN).

Detailed Description Text (44):

Although the media object server 220 and the client 100 output commands to the media server 130 generated by applications running on those platforms, those commands must be converted to device-level commands in order to control the particular media

server 130 receiving them. Accordingly, the commands are transferred to the media server 130 via a common API which is incorporated in the media manager 5. Although the media manager 5 is shown in FIG. 4 as being included in the media object server 220, the media manager 5 may be included in the media servers 130.

Detailed Description Text (45):

As noted above, the common application program interface allows the media object server 220 to interact with media servers 130 having different operational characteristics. The interface achieves this objective by translating the generic instructions or commands generated by applications running on the media object server 220 or client 100 into specific actions that are mapped to device-level commands which are output to a particular media server 130. Thus, the common application program interface ensures that applications need not account for the idiosyncrasies of a particular media server 130 in order to generate commands for that media server 130.

Detailed Description Text (46):

In addition, as noted above, applications can interact with each of the media servers 130 via the member functions of the CONTENT class in order to acquire information about objects stored within those media servers 130.

Detailed Description Text (47):

The above-mentioned individual APIs will usually be used in conjunction with a request, which is sent by the library client 100 to the library server 210, to store, retrieve, and update media objects. The storage, retrieval and updating of media objects within the multimedia data storage system is described in greater detail below.

Detailed Description Text (48):

When a request to retrieve or update a particular media object is generated by the library client 100, the library server 210 determines whether the media object is located within the multimedia data storage system by searching an index of media objects stored within the system.

Detailed Description Text (49):

Once the library server 210 determines that the media object is stored within the multimedia data storage system, it passes the request to that media object server 220 which retains control over the physical location of the media object. The media object may be stored in the media object server 220 itself, or in the media servers 130 or the media archive 140 associated with the media object server 220. Thereafter, the library server 210 may update the media object indexes and descriptive information stored in the library catalog 212 to which the library server 220 is connected to reflect the request.

Detailed Description Text (50):

After receiving the request to retrieve or update the media object, the media object server 220 identifies the specific type of media object (i.e. audio or video based file) based on an identifier included in the request. Thereafter, the object server 220 will identify whether the media object is located locally within the object store 222 or at a remote location within the media archive 140 or one of the media servers 130.

Detailed Description Text (51):

If the media object server 220 determines that the media object is located in the media archive 140, then depending on the type of the media object, the media object server 220 will output a command via the media manager 5 to the media archive 140 which instructs the media archive 140 to transfer the media object to the appropriate media server 130.

Detailed Description Text (52):

Thereafter, the media server 130 receiving the media object will output the same to the client media output station 110 upon receipt of a command to do so from the media object server 220 via the media manager 5. By way of example, the client media output station 110 may be a video display such as a television, broadcast monitor or a computer monitor including a CRT tube or flat panel display.

Detailed Description Text (53):

In an alternative scheme, once the media object is stored in the media server 130, the media object server 220 passes control information concerning the media object to the client 100, which then commands the media server 130 via the media manager 5 to output the media object to the client media output station 110. By way of example, the media object may be output as an asynchronous transfer mode (ATM) based video stream. In addition, the client media output station 110 may be separate from, or included within the client 100.

Detailed Description Text (54):

If the client 100 requests that a media object be stored or loaded into the multimedia data storage system, then the library server 210 catalogs indexing information concerning the media object. Subsequently, the library server 210 will request that the object server 220 store control information corresponding to the media object. Thereafter, the media object server 220 issues a command via the media manager 5 to the media server 130. In response to the command, the media server 130 will load the media object into itself from the client 100. The client 100 will then transfer the media object to the media server 130 completing the loading process.

Detailed Description Paragraph Table (1):

SERVER CLASS MEMBER FUNCTIONS CLOSE Closes an open media object and destroys a CONTENT Class instance. Parameters: An indicator of an instance of the CONTENT class. Return Codes: A first code indicates whether the operation was performed successfully. A second code indicates whether the media server reported an error. A third code indicates whether the operation failed. CLOSE\_SESSION Closes an open session. All resources allocated to the session will be released. Parameters: None Return Codes: A first code indicates that the operation was successfully completed. A second code indicates that the media server reported an error. A third code indicates that an invalid media group was specified. A fourth code indicates that the media server cannot be located. A fifth code indicates that the operation failed. COPY\_MEDIA Copies a media object from one media server to one or more other media servers. Due to the length of time that a copy operation may take, the COPY\_MEDIA member function method works asynchronously. That is, it is returned after initiating the copy operation. The progress of the copy operation may be determined by calling the GET\_MEDIA\_STATUS member function. The copy operation may be cancelled by calling the CLOSE member function. Parameters: An indicator of a media name. An indicator of a byte offset, at which to begin the copy operation. An indicator of a byte offset, at which to end copy operations. A zero value can be specified to indicate the end of the media object. An indicator of an array of MEDIA\_LOCATION structures. Each MEDIA\_LOCATION structure specifies one target location for a particular copy operation. The MEDIA\_LOCATION structure indicates the size of the structure, the number of users, the media server name, the media group name, and the media object name. A number of locations to which a media object will be copied. An indicator of a CONTENT instance pointer. Return Codes: A first code indicates that the operation was successfully performed. A second code indicates that the media server reported an error. A third code indicates that the media name is invalid. A fourth code indicates that an invalid media group (i.e. group media objects) has been specified. A fifth code indicates that a specified media name cannot be found. A sixth code indicates that a copy operation has failed. DELETE Removes an media object from the media server. Parameters: An indicator of a media name. A condition for deleting the media object. By way of example, this parameter may specify that the media object is to be deleted only if all of the resources are inactive. Alternatively, this parameter may mark the asset as unavailable, and defer deletion until media object is no longer in use. Return Codes: A first code indicates that the operation was performed successfully. A second code indicates that the media server reported an error. A third code indicates that the specified media name was not found. A fourth code indicates that the specified media object is in use. A fifth code indicates that the specified media name is invalid. A sixth code indicates that delete operation has failed. EVENT\_HANDLER Conditionally receives control for asynchronous events. An event mask, set with the REGISTER\_CALL\_BACK member function, allows specified events to be passed to the EVENT\_HANDLER member function. Parameters: An indicator of an event structure. By way of example, the event structure may specify the size of the event structure, an event type, a media event, and an error event. The media event structure specifies

an indicator of the CONTENT class, an event type, an index into a specific location or file array. The error event structure specifies the specific event associated with the occurrence of an error. Events are asynchronous messages sent to the media manager by the media server. Return Codes: This member function has no return type. GET\_EVENT\_MASK Returns the current event mask. Parameters: An indicator of an event mask. Return Codes: A first code indicates that the operation was successfully completed. A second code indicates that the event mask is not defined. GET\_META\_DATA Builds and returns a metadata file. Parameters: An indicator of a media name. A value which indicates whether playback should begin automatically, or delayed until play action. A size of user data. An indicator of a user defined data value. An indicator of a MEDIA\_STATUS structure to receive status information. The MEDIA\_STATUS structure specifies the size of the structure, the current length (bytes), the media copy rate, a media open mode, a creation date, the date of the most recent modification, and the date of the most recent access. An indicator of a buffer to receive metadata information. The size of buffer. Return Codes: A first code indicates whether the operation was successfully performed. A second code indicates that the media server reported an error. A third code indicates whether the specified media name is invalid. A fourth code indicates whether the specified media name is valid. A fifth code indicates that a specified media name was not found. A sixth code indicates that the operation has failed. A seventh code indicates that there is insufficient disk bandwidth for the operation. An eighth code indicates that an invalid media group has been specified. A ninth code indicates that the operation is not available on the media server specified. GET\_MEDIA\_STATUS Obtains the status of a media object. Parameters: An indicator of a media name. An indicator of a MEDIA\_STATUS structure which is used to receive status information. The MEDIA\_STATUS structure specifies the size of the structure, a current length in bytes of the media object, a media copy rate, a media open mode, the creation date of the media object, the last modification date of the media object, and the last access date of the media object. Return Codes: A first code indicates whether the operation was successfully completed. A second code indicates whether the media server reported an error. A third code indicates whether the specified media name is invalid. A fourth code indicates whether the operation has failed. INIT\_SERVER Initializes the Media Manager client library. This member function has to be the first call the application issues before requesting other Media Manager services. Parameters: The maximum number of sessions to be supported by the Media Manager. An indicator of a current version. Return Codes: A first code indicates whether the operation was performed successfully. A second code indicates that no more resources are available. A third code indicates that the operation has failed. LIST\_MEDIA\_GROUPS Obtains a list of configured media group names. Parameters: An indicator of an array of MEDIA\_GROUP\_ATTRIBUTES structures that will specify the media group name and type. A media group name is a variable-length string. The media group type includes default and current media groups. An indicator of the size in bytes of a group array. Return Codes: A first code indicates that the operation was successfully completed. A second code indicates that the media server has reported an error. A third code indicates that the media server cannot be located. A fourth code indicates that the operation has failed. LIST\_MEDIA\_NAMES Obtains a list of media names associated with a media group. Parameters: An indicator of a media group name. An indicator which specifies a buffer of media names. Each media name in the buffer is a variable length string. An indicator of the number of entries returned in the buffer. an indicator of the buffer size. An indicator of a position that is the start of a list of media names. Return Codes: A first code indicates that the operation was successfully completed. A second code indicates that the media server has reported an error. A third code indicates that the operation has failed. OPEN Returns an instance of the CONTENT class that is used in subsequent calls to read or write data from or to a media object. Parameters: An indicator of a media name. A mode of operation which corresponds to a read or write mode of operation. There are several options which can be specified along with this parameter. Namely, an indication can be provided of whether the media object is provided for exclusive use. An indication can also be provided that the media object should be created if it does not already exist. An indication can also be provided of whether content should be truncated from the media object. In addition, an indicator can be provided of the location of the end of the media object. The rate in bits per second that the media object will be read from or written to. A location of a pointer to an instance of the CONTENT class. Return Codes: A first code indicates that the operation was successfully completed. A second code indicates

that the media server has reported an error. A third code indicates that the specified media name was not found. A fourth code indicates that an invalid mode was specified. A fifth code indicates that an invalid copy-rate was specified. A sixth code indicates that a media object cannot be shared. A seventh code indicates that a request for exclusive use is denied, because a particular media object is in use. An eighth code indicates that a media name already exists. A ninth code indicates that the operation has failed. OPEN\_SESSION Opens a session from the application to the media server. Parameters:

#### Detailed Description Paragraph Table (2):

A host name of a media server with which to establish a session. A name of a media group. Return Codes: A first code indicates that the operation was successfully completed. A second code indicates that the media server has reported an error. A third code indicates that the operation has failed. A fourth code indicates that the media server cannot be located. A fifth code indicates that an invalid media group has been specified. A sixth code indicates that an invalid server name has been specified. REGISTER\_CALL\_BACK Registers the application event handler with the media manager. Once registered, the application may receive asynchronous event notification from the media server when necessary. Parameters: An event mask (i.e. bit mask) which is used to select which events are to be sent to the EVENT\_HANDLER method function. The event mask is developed using a logical operation involving the following event masks. A first event mask indicates a change in the state of a port. A second event mask indicates a change in the state of a media object. A third event mask indicates that an error has occurred. A fourth event mask indicates a change in the state of a media stream. A fourth event mask indicates a report of an event. Return Codes: A first code indicates that the operation was successfully completed. A second code indicates that the media server has reported an error. A third code indicates that an invalid event type has been specified. A fourth code indicates that the operation has failed. C\_SERVER Acts as a constructor which creates an instance of the SERVER class for a specific media server type. Parameters: A support level to be created within a SERVER class instance. Return Codes: None D\_SERVER Acts as a destructor with respect to an instance of the SERVER class. Parameters: None Return Codes: None UN\_REGISTER\_CALL\_BACK Unregisters an application event handler. When the event handler is unregistered, the application will not receive asynchronous event notifications. Parameters: none Return Codes: A first code indicates that the operation was successfully completed. A second code indicates that the media server has reported an error. A third code indicates that the session handler is corrupt. A fourth code indicates that the operation has failed. CONTENT CLASS MEMBER FUNCTIONS DESTAGE Transfers a media object from the media server to the media archive. This member function relies on other member functions. In particular, the member function calls the DELETE member function to delete a media object from the media archive if it exists, and then calls the COPY function to deliver the media object to the video archive. GET\_MEDIA\_ATTRIBUTE Obtains media attributes of a specified media object. Parameters: An indicator of a MEDIA\_ATTRIBUTES structure. The MEDIA\_ATTRIBUTES structure specifies the size of the structure, the play back rate in frames/sec, the play back rate in bits/sec, the expected number of users, the type of the media object, and the duration of the media object during playback. Return Codes: A first code indicates whether the operation was successfully completed. A second code indicates whether the media server reported an error. A third code indicates whether the operation has failed. GET\_MEDIA\_STATUS Obtains a status of a media object. Parameters: An indicator of a MEDIA\_STATUS structure to receive status information. The MEDIA\_STATUS structure specifies the size of this structure, the current length (bytes), the media copy rate, the media open mode, the creation date, the date the media object was last modified, the date the media object was last accessed. Return Codes: A first code indicates whether the operation was successfully completed. A second code indicates whether the media server reported an error. A third code indicates whether the operation has failed. LOAD Copies an existing media object from a client to a media Server. The media object must be open in write mode. This member function is an asynchronous function, i.e., it returns after initiating the load operation. The progress of this member function can be determined by calling the GET\_MEDIA\_STATUS member function. However the operation may be cancelled by calling the CLOSE member function. Parameters: An indicator of a MEDIA\_ATTRIBUTES structure. The MEDIA\_ATTRIBUTES structure defines the size of the structure, the type of the media object, the duration of the media object during playback, the playback rate in frames/sec, the playback rate in

bits/sec, the expected number of users, and the name of the media object. A host name from which the media object(s) are transferred. A user ID of the host specified by the hostname parameter. A password of the user ID specified by the user ID parameter. A file name array. All the files in the array can be joined together to form a single media object. This allows very large media objects to be created even if a particular media server does not support very large files. A number of file names in the file name array. An indicator of a MEDIA\_STATUS structure used to receive status information. The MEDIA\_STATUS specifies a size of the structure, the current length (bytes), the media object copy rate, a media object open mode, a creation date of the media object, the date that the media object was last modified, and the date that the media object was last accessed. Return Codes: A first code indicates whether the operation was successfully completed. A second code indicates whether the media server reported an error. A third code indicates that an invalid media server name has been specified. A fourth code indicates that an invalid server name has been specified. A fifth code indicates that the operation has failed. A sixth code indicates that there is insufficient space available in Media Server. A seventh code indicates that there is insufficient disk bandwidth in the media server. READ Reads data of a specified media object. Each successive call to the member function reads a sequential number of bytes from the current position. The media object must be open in read mode in order to call this member function. Parameters: An indicator of a pointer to which data is transferred. A number of bytes to be read. A number of bytes that have actually been read. Return Codes: A first code indicates whether the operation was successfully completed. A second code indicates whether the media server reported an error. A third code indicates that the operation has failed. RETREIVE Retrieves a media object from the media server. This member function calls other member functions. In particular, the COPY member function is called if a media object is not already stored in the media server. Subsequently, the GET\_META\_DATA member function is called to retrieve metadata information. SEEK Sets a current byte position within the media object. This member function only works in conjunction with the READ member function. The media object must be open in read mode in order for the operation to be initiated. It has no effect during write operation. Parameters: An indicator of how an offset parameter is used. The following support options may also be specified. A first support option sets a media byte position to the value of the offset parameter. A second support option sets the media byte position to its current position plus the offset parameter. A third support option sets the media byte position to the end of the media. Return Codes: A first code indicates whether the operation was successfully completed. A second code indicates whether the media server reported an error. A third code indicates that the operation has failed. SET\_MEDIA\_ATTRIBUTE Sets the media attributes of a media object. The media object must be opened in write mode to employ this member function. Parameters: An indicator of a MEDIA\_ATTRIBUTES structure. The MEDIA\_ATTRIBUTES structure specifies the size of this structure, an attributes flag, the type of the media object, the duration of the media object, the playback rate in frames/sec, the playback rate in bits/sec, the expected number of users, and the name of the media object. The attributes flag is a bit mask built up using a combination of basic attribute flags. The basic attribute flags specify which values in the structure are being set. The basic attribute flags set the media object name, the media object type, the duration of the media object, the media object, frame rate, the media object bit rate, the expected number of users, and all the attributes of the media object. Return Codes: A first code indicates that the operation was successfully completed. A second code indicates whether the media server reported an error. A third code indicates that the operation has failed. WRITE Writes data to the media object. Each successive call to this member function writes a sequential number of bytes. The media must be in open write mode. Parameters: An indicator of a pointer from which data is written. A number of bytes to be written. A number of bytes that have actually been written. Return Codes: A first code indicates whether the operation was successfully completed. A second code indicates whether the media server reported an error. A third code indicates that the operation has failed.

#### Other Reference Publication (4):

Leah-Martin, Tom, "General Magic Integrates Magic Cap with Oracle Media Server", Oracle Magazine, Spring 1994, pp(4).\*

#### Other Reference Publication (5):

Gibbs et al., "Multimedia Servers", Oracle Magazine, Fall, 1993, pp(5).

CLAIMS:

1. A common application program interface (API) which converts high-level generic commands received from a computer application into device-level commands which are output to a plurality of media servers which store media objects, said common API comprising a plurality of individual APIs which each perform a specific function, wherein each of said media servers has an operating protocol which responds to said device-level commands, and wherein said plurality of individual APIs comprise first and second groups of individual APIs, the first group of individual APIs corresponding to a first group of member functions associated with a class defining objects which represent media servers, the second group of individual APIs corresponding to a second group of member functions associated with a class defining objects which represent a logical description of a physical format of media objects.

4. The common application program interface defined by claim 1, wherein the first group of individual APIs comprises a COPY\_MEDIA API which copies a media object from one of the media servers to another one of the media servers.

5. The common application program interface defined by claim 1, wherein the first group of individual APIs comprises a DELETE API which removes a media object from one of the plurality of media servers.

12. The common application program interface defined by claim 1, wherein the first group of individual APIs comprises an INIT\_SERVER API which initializes a client library of a media manager containing the common API.

13. The common application program interface defined by claim 1, wherein the first group of individual APIs comprises a LIST\_MEDIA\_GROUPS API which obtains a list of names corresponding to configured groups of media objects.

14. The common application program interface defined by claim 1, wherein the first group of individual APIs comprises a LIST\_MEDIA\_NAMES API which obtains a list of the names of media objects associated with a particular media group of media objects.

16. The common application program interface defined by claim 1, wherein the first group of individual APIs comprises an OPEN\_SESSION API which opens a session to one of the media servers.

17. The common application program interface defined by claim 1, wherein the first group of individual APIs comprises a C\_SERVER API which is a constructor used to create an instance of the first class for a type of media server corresponding to at least one of the media servers.

18. The common application program interface defined by claim 1, wherein the first group of individual APIs comprises a D\_SERVER API which is a default destructor used to destroy an instance of the first class for a specific server type of media server corresponding to at least one of the media servers.

19. The common application program interface defined by claim 1, wherein the second group of individual APIs comprises a DESTAGE API which transfers a media object from one of the media servers to a media archive.

24. The common application program interface defined by claim 1, wherein the second group of individual APIs comprises a RETRIEVE API which retrieves a media object from one of the media servers.